Ser. No. 10/668,952

IN THE UNITED STATES
PATENT AND TRADEMARK OFFICE

Patent Application

Inventor(s)	Dong Lin
Case	1
Serial No.	10/668,952 Group Art Unit 2134
Filing Date	September 23, 2003
Examiner	Jacob Lipman
Title	Method And Apparatus For Defending Against Distributed Denial Of Service Attacks On TCP Servers By TCP Stateless Hogs

COMMISSIONER FOR PATENTS
P.O. Box 1450
ALEXANDRIA, VA 22313-1450

SIR:

DECLARATION UNDER 37 C.F.R. 1.131

1. I, Dong Lin, received a Ph.D. degree in Computer Science from Harvard University in 1998. I was a member of the technical staff at Bell Laboratories, Lucent Technologies Inc, Murray Hill NJ, between approximately 1998 and 2004, where I was doing research on, *inter alia*, network security.

I, the Declarant, Dong Lin, state further that:

2. I was the sole inventor of the present invention as recited in claims 1-16 of the instant application.

3. Prior to June 4, 2003 (the "critical date"), I conceived and reduced to practice, in the United States, the invention claimed in the instant application.

4. Attached as Exhibit 1 hereto is a copy of a (10 page) paper entitled "Defense Against Distributed Denial of Service Attacks" (hereinafter, "the Paper"), authored by me, the inventor of the instant application, which paper was completed and submitted for publication to the ACM SIGCOMM 2003 Conference prior to the critical date of June 4, 2003. (ACM SIGCOMM is the Association for Computing Machinery's Special Interest Group on Data Communication.) As can be seen by the heading of the Paper, it was submitted to ACM SIGCOMM 2003 having paper ID # 453. This fact is further evidenced by an internal corporate document, attached as Exhibit 2 hereto, entitled "Request for Approval of Manuscript", which requests a Lucent Technologies' patent attorney, Kenneth Brown, to approve the release of the Paper. As can be seen on its face, this document is dated prior to the critical date and was signed and dated by the Lucent Technologies' patent attorney, Kenneth Brown, prior to the critical date.

5. In addition, attached hereto as Exhibit 3 is a three page document entitled "Disclosure of Invention," signed and dated by me, Dong Lin, prior to the critical date of June 4, 2003. This document was submitted to the Lucent Technologies, Inc. Intellectual Property Law department for consideration of its merits as a possible basis for the filing of a United States patent application. Also, attached hereto as Exhibit 4 is a copy of a letter sent to me in response thereto from Kenneth Brown, the aforementioned patent attorney in the Lucent Technologies, Inc. Intellectual Property Law department, indicating that my submitted invention was being evaluated to consider its patentability. A patent application based on the submitted invention was in fact filed on September 23, 2003 (*i.e.*, the instant application, Ser. No. 10/668,952), after the aforementioned evaluation was completed and after a series of drafts of said application were prepared, reviewed and edited.

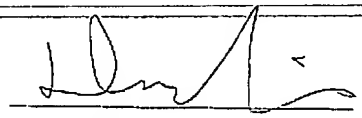
6. The invention claimed in the instant application is fully disclosed in the Paper ("Defense Against Distributed Denial of Service Attacks"). Specifically, for example, on page 4 of the Paper, section 4.1 thereof describes the invention as claimed, for example, by claim 1 of the instant application. In particular, section 4.1, in the first paragraph thereof, describes that the author "present[s] an algorithm that strengthens the TCP keep-alive mechanism and effectively detects and defeats a TCP stateless hog attack." In the third paragraph of Section 4.1, it explains that "an 'invalid' number be used as the sequence number in the keep-alive probe packets." Moreover, it

explains that, since "any fixed number could potentially be guessed . . . the choice should be random."

7. The Paper ("Defense Against Distributed Denial of Service Attacks") also shows that the instant invention was reduced to practice prior to the critical date. Specifically, for example, in the first paragraph on page 5 of the Paper (still in section 4.1), the above description of the inventive concept is immediately followed by an explanation that "[w]e have tested and confirmed our idea against TCP stacks in Microsoft Windows, Linux, FreeBSD, and OSF." Moreover, in Section 6 of the Paper on pages 6-8 thereof, additional information is provided regarding implementation and experimentation.

8. I certify that all statements made of my own knowledge are true and that all statements made on information and belief are believed to be true. I also understand that willful false statements and the like are punishable by fine, imprisonment or both under 18 U.S.C. 1001 and that willful false statements and the like may jeopardize the validity of the application-at-issue or any patent issuing thereon.

Date: December 19, 2007



Dong Lin

EXHIBIT 1

Defense Against Distributed Denial of Service Attacks

Paper Number: 453
Page Count: 10

Abstract

A Distributed Denial of Service (DDoS) Attack is a coordinated network attack scheme to deprive the victim's critical network resources and to cause service disruption. Such resources include the victim's network link to the outside world and its network servers' memory, CPU, and disks. Previous work on the defense against DDoS include IP trace back, firewalls, and more robust implementations of server operating systems.

This paper presents a set of DDoS defense algorithms that can be efficiently implemented at network edges to reduce or eliminate the impact of bandwidth and state attacks on TCP servers.

Our single-point-deployment approach avoids the requirement for coordination from other administration domains. It detects attacks in progress and reduces or eliminates the impact on the victim's network. While previous approaches work better with Internet wide ip-spoofing avoidance, spoofed packets have little impact on our system. The number of operations for stateful packet inspection on incoming packets is constant and therefore, is independent of the number of TCP connections sharing the link. This enables efficient software and hardware implementations.

An unoptimized kernel implementation can forward packets at approximately 1 gbps of aggregate TCP traffic without saturating an 866 MHz x86 CPU.

1. Introduction

Denial of service attacks cause service disruptions when limited server resources are allocated to the attackers instead of legitimate users. A distributed denial of service attack launches a coordinated DoS attack toward the victim from geographically diverse Internet nodes. The attacking machines are usually compromised zombie machines controlled by remote masters. The resources under attack include link bandwidth, server memory and CPU time. Distributed DoS attacks are more potent because of the aggregate effects of converging traffic, especially when the attackers have inside knowledge of the network topology.

TCP SYN flood [9], smurf IP ping [10], and bandwidth attacks on root name servers [30,8] are examples of DDoS attacks previously deployed. In [22], a trace study has revealed that there have been far more attacks than previously known.

There are numerous studies on improving server operating systems to resist resource exhaustion [2,4,5,18,21,27]. Several authors have investigated better network protocol design principles to protect servers from attacks on stateful handshake protocols [1,16,17]. IP trace back is a network wide coordinated effort to follow the packets back to their originators [3,7,11,24,26,29]. A lot of attacks could have been prevented entirely if ingress filtering were deployed Internet wide [12] or a source address validation protocol were widely used [19]. Heuristic-based bandwidth attack detection by signatures were studied in [14,20] for monitoring and network congestion control. In [13], the authors proposed classifying incoming packets into UDP, TCP, TCP SYN, and CGI categories and enforcing class-based rate control and weighted fair queueing.

DDoS attack tools tend to mutate and evolve over time. With wider deployment of egress filtering, attackers will exploit doors that are most likely to be left open (e.g., TCP, DNS). Attack signatures will change or disappear to evade detection. We believe that sophisticated future attacks will be almost indistinguishable from legitimate ones. Filtering alone is not only inefficient but also insufficient. Lots of false positives will force researchers to go back to the drawing board for new heuristics.

A one-for-all solution to DDoS defense will not exist. A more practical approach is to raise the bar higher so that the attackers will have to commit as much resources as the victims. A simple but expensive solution adopted by many large corporations is to buy the highest ingress link possible so that bandwidth attacks would be difficult without driving a significant portion of the Internet at core speed.

An alternative to the expensive upgrade is promoting and adopting DDoS resistant network protocols, as suggested by the authors of [17]. Following the philosophy of making DDoS attacks more resource bound, we present a network edge approach to the defense. The fundamental principle is to force attackers to commit as many resources as legitimate users in order to tie up equivalent amount of bandwidth. This scheme is particularly suitable to protecting online service providers' networks where a large user population is typical. This means legitimate users would have to be vastly out numbered in order to bring down the entire edge link to the victim's network.

This paper makes the following contributions to the research of distributed denial of service defense:

- We present a single-point queueing and buffer management scheme as an effective mechanism to defend against bandwidth attacks.
- We describe a unified approach to prevent TCP SYN attacks on both state and bandwidth.
- We propose a TCP enhancement to defeat stateless TCP state hoggers.

It's a single-point deployment gateway scheme, so that network wide coordination and modifications to the current network infrastructure are unnecessary. The deterministic scheme eliminates any false positives. Unlike any filtering approach, our efficient data-structures and algorithms require constant number of operations per packet. This makes efficient software and hardware implementation feasible.

The core of the scheme is the new application of edge queueing as a buffer management mechanism to provide isolation among different protocols and isolation among individual TCP connections sharing the link. This single-point queueing approach guarantees max-min fairness to legitimate users and our efficient manipulations of the data structure always put the attackers on top of the suspects list. Inspired by previous studies, we propose a backward compatible TCP enhancement to provide client puzzles and detect state hoggers.

For the rest of the paper, we use TCP flows and connections interchangeably to refer to the association of TCP packets identified by their source and destination addresses and port numbers.

2. DDoS Gateway Architecture

This section gives an overview of the design of our DDoS gateway. The goal of the research is to prevent attacks on a network of TCP services such as HTTP, FTP, SMTP, IMAP, and many others. Non-TCP services like DNS can also be present, but we assume that such traffic contributes to an insignificant fraction of the link usage under normal circumstances. These packets are isolated upon arrival at the network edge gateway and are given minimal fraction of the link capacity at the presence of TCP traffic. As explained later in this section, our DDos gateway isolates TCP traffic from other protocols so that attackers would have to use TCP packets in order to disrupt TCP services. In Section 7, we give a brief discussion on dealing with non-TCP traffic.

We further assume that future DDoS attacks will be more sophisticated and almost indistinguishable from legitimate ones. Imagine the attacker initiating a million simultaneous well-behaved TCP connections toward a single web server. A complete prevention solution will be costly, if not impossible. Instead, our DDoS gateway attempts to make the attacks themselves more resource bound and therefore more costly for the attackers. It will stop the attacker from launching a massive attack utilizing only a handful of computers. Many of the previous attacks belong to this category. A SYN flood attack can overwhelm a server with a small stream of SYN packets [9]. A smurf ICMP ping packet can multiply its impact by a factor of hundreds through IP broadcast [10]. Our approach would force the attacker to use TCP packets only and they have to vastly outnumber legitimate TCP connections at a TCP service network. Such attacks should be expensive and difficult to accomplish with limited computers and access points in the network.

As depicted in Figure 1, our DDoS gateway is positioned at network edges, one hop upstream from the protected link. A network edge is located at the border between an end-customer and its provider's network. Packets from the network core arrive from a link several times faster than the exit links leading to end-customers. They are then demultiplexed to various end networks. End network links can be flooded during peak usage or under a denial of service attack in-progress.

At our DDoS gateway, packets arriving from the core are first dispatched to isolate TCP packets from others. Data packets with prior states at the gateway are sorted into various queues, subject to certain buffer management policies. SYN packets do not require prior states to be forwarded and are handled separately by the Connection module. Data packets, SYN packets, and packets from non-TCP protocols are then scheduled to appear on the protected link. For the reverse direction, packets from the TCP servers are examined and provide stateful information for further handling of incoming packets from the core. A TCP stateless hogger detection mechanism is built into the Watch module.

The rest of the paper is organized as follows. Section 3 describes connection request handling. Section 4 describes a form of state attack on the gateway and the TCP servers behind it. We describe a solution implemented in the Watch module. Section 5 details packet queueing, scheduling, and buffer management in the FlowQ module. In Section 6, we provide details of our practical implementation and experiments.

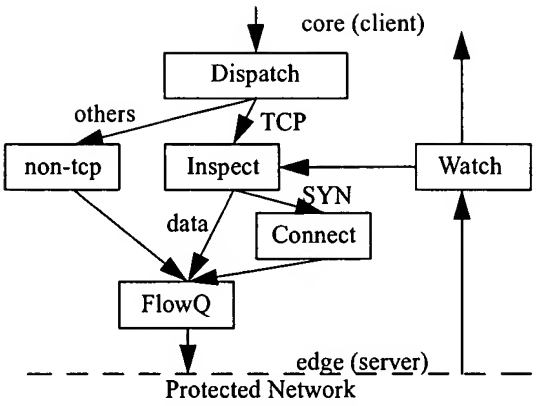


Figure 1. Functional diagrams of the DDoS Gateway

3. TCP SYN Flood Prevention

Every TCP connection starts with a SYN packet. TCP servers must respond to every valid SYN request with a SYN,ACK and retransmit it if necessary. SYN packets penetrate transient firewalls without prior states in them. They also cause servers and firewalls to allocate resources in preparation for new connections. As a result, they are the first potential vehicle for launching attacks. This section describes two forms of such attacks, one of which has been studied previously.

3.1 SYN State Attacks

TCP connections are established via a three-way handshake. Incomplete connections are usually held in a per listener queue. The limit on the size of the backlog queue is small. A malicious user can overwhelm a TCP server by sending connection request SYN packets without completing the rest of the handshake. The backlog queue on the server eventually overflows, causing denial of service to legitimate requests.

This is an example of a state attack on TCP servers. A TCP handshake is the way for the server and the client to exchange and agree on certain parameters to be used for the data transfer. Such parameters include maximum segment size, window scales, time stamps, and others. The server has to store such information before completing the handshake. Two solutions exist to defend against SYN attacks which intend to deplete the server's memory pool. One is to reduce the amount of memory used for incomplete connections. The other is to eliminate any memory usage entirely.

The SYN Cookie approach stores the negotiated parameters in the subsequent SYN,ACK and ACK packets, avoiding any memory allocation for incomplete connections [4]. This is done by using a cryptographic hash as the server's initial sequence number. A connection's parameters are encoded in the 32-bit hash. See [18] for a brief discussion of SYN cookie's incompatibility issues with TCP extensions, SYN,ACK retransmission, and T/TCP.

SYN Cache replaces the tiny per socket backlog queue with a global pool of kernel buffers [5,18]. The size of each cache entry is much smaller than those for established connections. The cache is organized as a hash table indexed by a hash of a connection's addresses and port numbers. Each bucket has a preset upper limit. The total amount of memory allowed for the SYN Cache is determined by the per-bucket limit and the total cache size. When a bucket overflows, the oldest entry in the same bucket is replaced. When the cache overflows, the oldest entry in the whole cache is replaced. Notice that SYN Cache only *defers* the SYN attack problem by raising the bar. But it seems to be an efficient way of using kernel resources and does not have compatibility issues that exist in the SYN Cookie approach. The default FreeBSD 4.7 kernel uses a cache limit of 16K entries and a bucket size of 30. By default, FreeBSD also uses SYN Cookie to recover connections evicted from the SYN Cache.

3.2 SYN Bandwidth Attacks

While TCP servers can defend against SYN state attacks by themselves, bandwidth attacks using SYN packets are almost impossible to deal with at downstream. A typical SYN packet is no more than 64 bytes long. A burst of such minimum size packets can cause livelock on the server [21]. Our laboratory experiments showed that an 866 Mhz x86 PC, which is capable of driving a TCP connection at 1 gigabits per second with 1518 byte packets, can only receive 64 byte packets at 200 mbps without dropping. A deadly attack would consist of blasting the server's ingress link with small packets. Many optimizations exist to avoid receiver livelocks [2,27]. In general, bandwidth attacks must be dealt with further upstream, before the damage is done.

SYN packets are special in that they do not require prior states in order to penetrate the edge network defense. These stateless packets must be isolated and separated from others at upstream. This section focus on the handling of SYN packets, we defer our study on regular TCP data packets over established connections until section 5.

In our DDoS gateway, incoming SYN packets are queued separately from other TCP packets. The egress packet scheduler gives a fair share to each of the non-empty queues. We describe the details of the scheduler in section 5. The scheduling policy ensures that SYN packets cannot overwhelm the egress link in the presence of other packets. Another advantage of such scheduling scheme is that it interleaves 64 byte packets with data packets that are usually bigger, reducing the likelihood of causing livelock at the TCP servers.

Instead of storing and passing through SYN packets blindly, exponential gaps between adjacent SYN packets

from the same connection are enforced. Like other retransmissions, TCP protocol requires that retransmissions for SYN packets must follow the same exponential back off time intervals for data packets. With an initial interval of 500 milliseconds, subsequent SYN packets arriving before the current interval time are dropped. Incomplete connections that have lasted over a extended period of time are removed if they are not evicted by new connection requests by then. The system further requires that each incomplete connection has no more than one SYN packet in the queue. This eliminates the possibility that the egress link be flooded by a small number of incomplete connections.

In order to prevent randomly generated SYN packets from creating a large number of states in the gateway and flooding the egress link, the total number of incomplete connections allowed in the system is in proportion to the number of currently established connections in the following way:

$$P = M + c * N \quad (1)$$

M and c are constants and N is the number of established connections in the system. The intuition is that legitimate TCP connection requests would almost certainly turn into established ones. So the number of legitimate requests and the number of established connections should be closely correlated. The constant c is two in our prototype system. M is the number of requests allowed with no active connections in the system for head start.

When a new SYN packet arrives without prior state, a new state is allocated if condition (1) holds. Otherwise, a randomly chosen state is evicted. We prefer random selection over aging because it's more friendly to connections with long round-trip times when the system is under attack. A connection is moved from the incomplete state to established state when a valid returning ACK packet passes through. The time taken for the transition is largely dependent on the round-trip time between the server and the client. If a retransmitted SYN packet arrives, the exponential interval gap is checked.

Accepted packets are enqueued into a FIFO buffer. If a packet buffer is unavailable, the packet buffer manager removes a packet from the system. See Section 5 for details on buffer management.

An alternative to the above approach is to have a TCP handshake proxy which forwards the initial SYN packet and all of the server's responses. The proxy performs SYN retransmission on behalf of a client if necessary until the client returns a valid ACK or a connection time-out occurs. This proxy approach works equally well, but it requires more connection states to be stored at the gateway for SYN retransmission and is more expensive to implement.

4. TCP Stateless Hog Elimination

Each established TCP connection creates a state on a TCP server. The amount of allocated memory for storing these states is proportional to the total number of open connections. A denial-of-service attack on a TCP server can be launched by continuously creating new TCP connections with the targeted server until it runs out of memory and unable to accept service requests from legitimate users.

There is no known defense against such DoS attack. Section 4.2.3.6 in RFC 1122 [6] addresses a related issue. A TCP keep-alive mechanism is invoked in server applications that might otherwise hang indefinitely and consume resources unnecessarily if a client crashes or aborts a connection during a network failure. This mechanism could be used to alleviate the TCP server during an attack in-progress.

However, the keep-alive mechanism can be easily defeated without much cost for the attacker. The attacking machines, usually victims of break-ins themselves, respond to keep-alive probes and effectively prevent the server from closing the connection. To do that, the attacker simply crafts the returning ACK from the keep-alive packet, switching the fields of sequence number and acknowledge number. The memory required on the attacking host is constant and is unrelated to the number of TCP connections created for the attack. We name this kind of DoS attack a "TCP stateless hog."

4.1 The Detection and Prevention Algorithm

We present an algorithm that strengthens the TCP keep-alive mechanism and effectively detects and defeats a TCP stateless hog attack. It should eventually be adapted into every TCP implementation if possible. It can also be deployed at network edges in DDoS gateways to protect legacy systems.

Among other information, a keep-alive probe packet contains a sequence number corresponding to the latest octet from the server to the client. The client responds with an acknowledgment in agreement with the same octet. The attacker answers the probe by copying the sequence number.

We propose that an "invalid" number be used as the sequence number in the keep-alive probe packets. TCP's specification [23] requires that the acknowledgment segment contain the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received. Because TCP sequence numbers are 32 bit unsigned integers and do wrap around, any number could be valid. The most likely "invalid" one is the initial sequence number minus one. However, any fixed number could potentially be guessed if the attacker knows the algorithm. Therefore the choice should be random. In order to avoid confusion with numbers currently in use by the connection, the selection space should be as far away from the current sequence number as possible. It's quite unlikely that a TCP connection would exhaust the entire 32 bit integer space within one round-trip time. Our initial implementation is based on the following formula:

```
keep-alive.seq=snd_una-(1<<30)-random(1<<20)
```

The construction of such an acknowledgment requires maintaining states on the responding host as done in legitimate and conforming TCP implementations. TCP's specification [23] *requires* that the acknowledgment segment contains the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received. Therefore, all conforming TCP implementations are mandated to respond with the latest sequence number. We have tested and confirmed our idea against TCP stacks in Microsoft Windows, Linux, FreeBSD, and OSF. The only exception is an unknown TCP proxy managed by a wireless cellular phone operator. The connection state was removed after we sent out the modified keep-alive packet.

A TCP stateless hog program will produce an incorrect response that can be detected by the TCP server and the DDoS gateway. Connections associated with these wrong answers can be removed with 100% accuracy.

One might argue that forcing the attacker to allocate four bytes for each open connection is still not a big commitment. Connection closure is both policy and application dependent. A legitimate shell session may stay for days without exchanging any data. A login daemon should not keep an unauthenticated connection prompt open for more than a few seconds. A busy web server should keep all open connections utilized and short-lived. These responsibilities lie outside of the network.

5. FlowQ: Constant Time Selective Discard with Per-Flow Queueing

At network edges, ingress links from the network core are normally faster than egress links toward end customers' networks. Egress links can be flooded if ingress packet arrival rates are higher than the link capacity. It is important to note the fact that TCP is a congestion aware transport protocol which throttles back the packet rate upon detection of congestion. As a result, a malicious user can force other legitimate TCP connections to reduce their usage on the egress link without massively flooding the link. This phenomenon also occurs during the existence of connections driven by aggressive TCP implementations, malicious or not.

To protect legitimate TCP flows, fair scheduling and fair buffer management mechanisms are needed. This section describes the FlowQ module, a fair scheduling and fair dropping buffer management scheme.

TCP data packets arriving from the core network destined for the same output interface are first sorted into flow queues. A flow is a single TCP connection identified by its source destination addresses and port numbers. An egress packet scheduler manipulates these flow queues for ordered packet departure. For the rest of this section, we use the term port to represent a physical network interface. Packets arrive on input ports and depart on output ports. Our DDoS gateway uses an output port buffering architecture.

5.1 The Output Port Packet Policing Model

An output port per-flow queueing and buffer sharing architecture has two interfaces at each output port: arrival side and departure side. The arrival side may receive packets from the input ports at a rate higher than the output port line speed, causing packets to be buffered and delayed. The departure side serves multiple flows that have packets queued. As a result, an optimal buffer management policy should demonstrate two distinct properties:

- Fair scheduling at departure enforces that all competing flows are entitled to the same bandwidth resource.
- Fair dropping at arrival makes sure that no flows use more buffers at the expense of others.

With *unfair* scheduling, some flows can obtain higher bandwidth even if they do not use more buffers than others. With *unfair* buffer allocation, a flow might not have packets to send even if the scheduler is perfectly fair.

Given per-flow queueing, Round Robin scheduling has been shown to be max-min fair [15]. Bit by bit round-robin scheduling can be efficiently approximated to work with variable packet lengths [25]. A packet discard algorithm for per-flow queues is straightforward: accept all incoming packets as long as there are free buffers. When a packet arrives without any free buffers, drop a packet from a flow with the most bytes buffered[1].

To avoid unnecessary purging (in which an accepted packet is immediately replaced by the next arrival) and to prevent TCP flows from falling into time-outs due to tiny windows, the incoming packets should not be dropped when the longest queue has only two packets. This limits the maximum number of flows to be B/2, where B is the total buffer size committed to the system.

5.2 Implementing FlowQ with O(1) Complexity

Implementation of round robin scheduling is known [25]. However, the FlowQ buffer policy requires the system to keep track of flows with the most packets queued. In this section, we describe our design for sorting flows based on their queue lengths using a constant amount of work per packet.

For ease of presentation, assume for now that all packets are of the same size. In other words, assume that queue length is in the unit of packets as opposed to bytes. The system maintains a chain of lists. Each sublist contains all flows with identical queue lengths. These sublists are then sorted according to the queue length in descending

1. We ignore the possibility that the purged packet might be smaller than the incoming, in which case, either multiple packets have to be dropped or both packets are dropped. The latter is a non-optimal solution, whereas the former might not be efficiently implemented.

order. Initially, all lists are empty, implying all flows have empty queues.

When a packet for $flow_i$ arrives with free buffers available, it is accepted and appended to $flow_i$'s queue of length Q_i. The queue length increases to Q_i+1. The system removes $flow_i$ from its current sublist $SL(Q_i)$ and appends it to sublist $SL(Q_i+1)$. This sublist should be either one item away from $SL(Q_i)$ because they differ by *exactly* one packet, or does not exist if no flow has queue length of Q_i+1 before the arrival. In the latter case, the system simply creates a list and inserts it before $SL(Q_i)$. In both cases, no scanning of any list is needed in order to maintain the ordered master list. Figure 2 below depicts this enqueue operation.

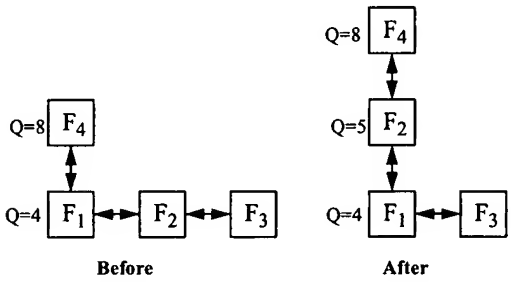


Figure 2. Packet queue sorting on arrival.
A packet arrival for $flow_2$ causes its removal from SL(Q=4) and insertion into SL(Q=5).

When a packet for $flow_i$ is scheduled and departs, it is dequeued from $flow_i$'s queue with length Q_i. The queue length decreases to Q_i-1. The system removes $flow_i$ from its current sublist $SL(Q_i)$ and prepends it to sublist $SL(Q_i-1)$ if $Q_i > 1$. $SL(Q_i-1)$ should be created if necessary. The old sublist $SL(Q_i)$ should be removed and deleted if empty. Once again, no linear scanning is needed. Figure 3 depicts this dequeue operation.

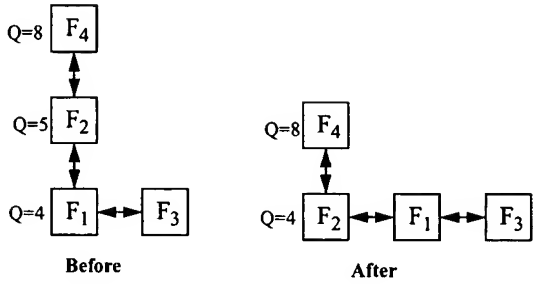


Figure 3. Packet queue sorting on departure.
A packet departure for $flow_2$ causes its removal from SL(Q=5) and insertion into SL(Q=4).

When a packet for $flow_i$ arrives on a full buffer, a $flow_j$ with the longest queue length can be found from the first sublist in the sorted master list. The system purges a packet from $flow_j$ if $i \neq j$ and accepts the incoming packet. Otherwise the arriving packet is dropped. Operations of purging a packet are identical to that of scheduling a packet.

If packets are of variable length, queue length has the unit of bytes and each sublist contains all flows with the same $[Q_i/MTU]$ value, where MTU is the maximum packet size and [] is the ceiling function.

Similar operations are performed on arrivals and departures. When $flow_i$ with Q_i is added to $SL([Q_i/MTU])$, it is appended if Q_i mod MTU < MTU/2 and prepended otherwise. This ensures the sorting error within the sublist is less than MTU/2. It might be possible that an arrival or a departure of a small packet does not cause its flow to be moved to a neighboring sublist. Instead, it's prepended or appended to the same sublist depending on whether the updated queue length crosses the MTU/2 boundary.

6. Implementation and Experiments

In the original design of the FlowQ module, a TCP flow needs to be shuffled back and forth among sublists $SL(Q_i)$ for each arrival and departure. In order to reduce the number of memory operations in actual implementations, we do the shuffling at a coarser level. Each sublist SL_k contains flows with queue lengths satisfying $k = [Q/B]$, where B is a constant. We set B to 16 in our system. This way a flow only needs to be shuffled if the queue length changes by more than 16 packets. The number of operations is reduced at the expense of precision. However, we append a flow to a sublist for an upward shuffle, and prepend a flow for a downward shuffle. This has the effect of putting flows with longer queues at the upper half of each sublist. When a buffer needs to be evicted, we dequeue one from the top sublist.

In our initial implementaion, we also used packets as the unit for queue length. This imposes strong bias against flows with large number of small packets. A well behaved TCP connection sends out small packets only if it has no more data to send, i.e. the number of in-flight packets is small. A typical application is login. We do not expect such connections to have more than two packets in the queue. So the packet based queue accounting mechanism should not discriminate them in any way. This biased accounting targets connections generating lots of minimum sized packets, intending to cause livelock on the servers.

A preliminary implementation was done under FreeBSD 4.7. The kernel is unmodified except that ethernet device drivers interact with our DDoS module instead of the generic kernel ethernet layer. It is unoptimized in that packet arrivals and departures are still interrupt driven as opposed to polling based. On an 886 Mhz x86 CPU with NetGear GA620 and Intel Pro/1000 gigabit adaptors, the system is able to forward multi-flow TCP packets at 500 mbps. Each packet appears on the 64-bit 66 Mhz PCI bus twice, once for arrival and once for departure. The total aggregate bandwidth over the PCI bus is around 1 gbps. The same machine can drive a single TCP flow over a single gigabit card at 1 gbps with almost no idle cycles. Each device is configured to handle 1500 byte maximum segments and performs hardware TCP/UDP/IP checksum. These numbers suggest that our DDoS module introduces little processing overhead to the forwarding engine.

Figure 4 shows the test equipment configuration used to perform the experiments in this section. Fifteen PCs, M1 through M15, connect to a switch over 100 mbps fast ethernet links. The DDoS gateway connects to the switch via a gigabit fiber. The protected link between the DDoS gateway and PC G1 also runs at 100 mbps. All experiments involve machines M1 to M15 generating traffic toward machine G1 through the DDoS gateway.

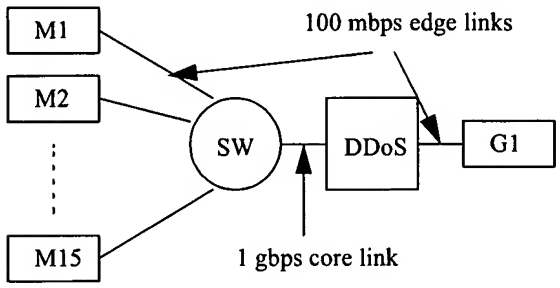


Figure 4. Test Equipment Configuration
Fifteen PCs, M1 through M15, connect to a switch over 100 mbps links. The DDoS gateway connects to the switch via a gigabit fiber. The protected link between DDoS and PC G1 also runs at 100 mbps.

The first experiment demonstrates DDoS gateway's handling of mixed UDP and TCP traffic. A UDP DNS blast program starts from machine M1 toward machine G1 at full line rate. 60 seconds later, three infinite transfer TCP connections from machines M2, M3, and M4 start at 10 second intervals toward machine G1. Figure 5 shows each connection's observed bandwidth over the edge link. Bandwidth samples are taken at one second intervals.

In the presence of TCP packets, the FlowQ module builds up a queue for each connection. Packets are scheduled out on the edge link in a round-robin fashion, causing the single UDP queue to grow faster than others. Eventually limited packet buffers are exhausted, new arrivals cause packets to be removed from the UDP queue. Packet losses only occur on the longest queue. There are no TCP retransmission time-outs.

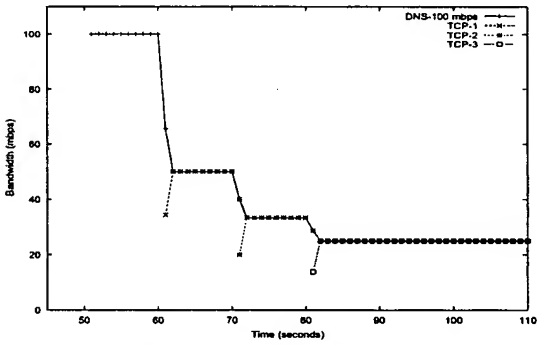


Figure 5. Protecting TCP traffic from UDP blast
DDoS reduces UDP bandwidth at the increasing presence of TCP flows.

The next experiment involves two TCP connections driven by different stacks, good-tcp and bad-tcp. Both are full BSD stacks except that bad-tcp sends N copies of each packet. Machine M1 runs the good-tcp stack, M2 operates the bad-tcp stack. Data packets go from the M machines to G1. Table 1 shows the observed TCP goodput on G1 with the DDoS gateway in bridge mode (i.e. no per-flow queueing, no scheduling, just a simple FIFO). In all cases, the good-tcp and the bad-tcp obtained the same goodput. However, the bad-tcp consumed N times the bandwidth at the expenses of the good-tcp. Because both stacks have the TCP congestion aware mechanisms, the gateway was not flooded with too many packets.

	N=1	N=2	N=4	N=8
good-tcp	47	31	17	9
bad-tcp	47	31	17	9

Table 1. good-tcp and bad-tcp stacks compete over the edge link without DDoS protection. Numbers are TCP goodput in mbps.

We were surprised to see the numbers in Table 1 unchanged when we turned the DDoS gateway into full protection mode. Packets from the two flow were queued

separately and scheduled fairly, but the good-tcp could not run beyond more than 1/(N+1) of the link bandwidth. Frustrated in disbelief, we resorted to simulate the exact configuration and exact TCP stacks in a network simulator. The numbers were different and more expected.

Further investigation of the actual experiment showed that the instant queue length ratio at the gateway is usually N:1 in favor of the bad-tcp connection. A typical sign of FIFO queueing effect. Furthermore, queue lengths were short and mostly stayed at 2N and 2 packets. There was no packet loss. The sender's congestion windows were maxed out at 64KB, a FreeBSD default. With the maximum segment size set at 1460 bytes, these numbers suggested that 45(N+1) packets worth of buffering existed somewhere in the system. But we only saw 2N+2 in the DDoS gateway, plus two buffers in the kernel send queue between the OS and the device driver. We finally realized that most ethernet adaptors have transmit ring buffers on local memory. The NetGear GA 620 gigabit card (running at reduced speed of 100 mbps) we used has a default buffer size of 512 packets! This of course was not in the simulator. After reducing this buffer to 32 packets, the numbers were much up to our expectations, as shown in Table 2. The good-tcp stack was able to maintain half of the link bandwidth, with DDoS protection. The bad-tcp got the other half, but it's goodput is only 1/N th of that due to the wasted bandwidth on duplicates. Imagine that if the number of connections operated by good-tcp stacks increases to one million, the behavior of the bad-tcp stack almost has no impact on the operation of the link. The attacker would have to run one million flows across in order to take away half of the link capacity.

	N=1	N=2	N=4	N=8
good-tcp	47	47	47	47
bad-tcp	47	23	12	6

Table 2. good-tcp and bad-tcp stacks compete over the edge link with DDoS protection. Numbers are TCP goodput in mbps.

A lesson learned from this experiment is that a malicious tcp could evade detection if significant FIFO effect exists between the policy enforcement mechanism and the transmission media. The motivation for the use of local adaptor buffers is efficiency for high-speed media. It vastly reduces the possibility of buffer underflow and can also reduce the number of transmit interrupts posted to the CPU. To alleviate the adaptor buffer FIFO effect without compromising efficiency, we are investigating a change to our system. Flow queue lengths are decremented only after the packets get actually transmitted over the wire. This can be done when the device driver recycles the transmit buffers. We further impose a per-flow buffer limit to prevent the entire adaptor's local buffer to be overwhelmed by a small number of flows. Packets are dropped if queue lengths exceed the limit. This should signal TCP to start throttling transmission earlier and more effectively. A limit of 12 should be big enough for the majority of existing TCP implementations to recover packet losses without going through retransmission time-outs.

The consequence of over buffering is the increased end-to-end delay. During the experiment, ping delay was as high as 50 milliseconds with N=8. We have actually observed delays of over a second behind DSL and cable-modem lines without any packet loss, a sign of overbuffering in today's network.

7. Dealing with Non-TCP Traffic

In our current design, non-TCP packets are sorted into a single FIFO queue. The scheduler treats them as if they belonged to a single TCP connection. We are investigating refined schemes as future work. The treatment of DNS packets are the focus of our study.

A typical DNS transaction can finish in one round-trip time with two packets. The connectionless nature of DNS makes packet classification difficult. One of our resource allocation metrics, condition (1), no longer exists in DNS. Random drop on the FIFO queue provides better protection than tail drop. But the advantage is minimal.

There are strong arguments against adapting TCP for DNS. It requires at least seven (7) packets and two round-trip times for each transaction. The requirement for additional resources is not the concern here. But the benefits for adapting TCP are also strong:

- A majority of the attacking SYN packets would have been dropped at the DDoS gateway.
- Spoofing is no issue for TCP.
- The DNS server would not have been fooled to bombard the reverse link easily (DNS replies can be ten times bigger than the queries themselves).
- The attacking machines are traceable once the TCP handshake completes.

Other options for protecting root name servers do exist. But for networks that do have to run their own DNS servers behind a single edge link, this remains a problem for further study.

8. Conclusions

Following the philosophy of making DDoS attacks more resource bound, we present a network edge approach to the defense. The fundamental design principle is that the attackers would have to allocate as much resources as other legitimate users in order to take away an equal amount of the victim's link bandwidth allocated for legitimate usage. This scheme is particularly suitable to protecting online service providers' networks where a large user population is typical. This means legitimate connections would have to be vastly out numbered in order to bring down the entire edge link to the victim.

It's a single-point deployment gateway scheme, so that network wide coordination and modifications to the

current network infrastructure are unnecessary. The deterministic scheme eliminates any false positives. Unlike any filtering approach, our efficient data-structures and algorithms require constant number of operations per packet. This makes efficient software and hardware implementation feasible.

Our queueing approach guarantees max-min fairness to legitimate users and our efficient manipulations of the data structure always put the attackers on top of the suspect list. Per-flow queueing schemes have been proven to provide fairness and isolation. Counter arguments have been focused on its requirement of end-to-end and hop-by-hop deployment. Our single-point deployment demonstrates a new application of per-flow queueing to the defense against distributed denial of service attacks.

Inspired by previous studies, we propose a backward compatible TCP enhancement to provide client puzzles and detect system state hoggers. Some of the techniques can be adopted to server operating systems. Legacy systems benefit from a single-point deployment at the gateway.

Our DDoS gateway currently only protects the edge link against attacks from the core network. The authors in [24] described ways for a malicious TCP receiver to fool TCP servers to attack their own outgoing link from the opposite direction. Some of the attacks can be avoided by patching TCP implementations. Others are left as open issues. We believe that per-flow queueing from the other side of the link is a more general approach to alleviate the problem.

9. References

[1] Aura, T., Nikander, P., and Leiwo, J., "DoS-resistant authentication with client puzzles," Proceedings of the 8th International Workshop on Security Protocols, April, 2000.

[2] Banga, G., Druschel, P., Mogul, J., "Resource Containers: A New Facility for Resource Management in Server Systems," Proceedings of the 1999 USENIX/ ACM Symposium on Operating System Design and Implementation, pp 45-58, February 1999.

[3] Bellovin, S. M., "ICMP traceback messages," Internet draft, http://www.ietf.org/internet-drafts/draft-ietf-itrace-03.txt

[4] Bernstein, D.J. SYN cookies. http://cr.yp.to/syncookies.html

[5] Borman, D. BSDI Implementation of SYN Cache. ftp://ftp.bsdi.com/private/44-syn-diffs.gz

[6] Braden, R., "Requirements for Internet Hosts -- Communication Layers," RFC 1122.

[7] Burch, H. and W. Cheswick, "Tracing Anonymous Packets to Their Approximate Source," Proceedings of 2000 Systems Administration Conference, December 2000.

[8] CAIDA, "Nameserver DoS Attack October 2002," http://www.caida.org/projects/dns-analysis/ oct02dos.xml

[9] CERT Coordination Center, "TCP SYN Flooding and IP Spoofing Attacks," http://www.cert.org/advisories/ CA-1996-21.html

[10] CERT Coordination Center, "Smurf IP Denial-of-Service Attacks," http://www.cert.org/advisories/CA-1998-01.html

[11] Dean, D., Franklin M., and Stubblefield, A., "An algebraic approach to IP traceback," Proceedings of the 2001 Network and Distributed System Security Symposium, February 2001.

[12] Ferguson, P. and Senie, D., "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," RFC 2267, January 1998.

[13] A. Garg and A. L. Narasimha Reddy, "Mitigation of DOS attacks through QOS regulation", in Proc. of IWQOS workshop, May 2002.

[14] Gil, T.M. and Poleto, M., "MULTTOPS: a data-structure for bandwidth attack detection," Proceedings of 10th USENIX Security Symposium, August 2001

[15] Hahne, E., Gallager, R., "Round Robin Scheduling for Fair Flow Control in Data Communications Networks," IEEE International Conference on Communications, June 1986

[16] Juels, A., and Brainard J., "Client puzzles: A cryptographic countermeasure against connection depletion attacks," Proceedings of the 1999 Networks and Distributed System Security Symposium, March 1999.

[17] Leiwo J., Nikander, P., and Aura, T., "Towards network denial of service resistant protocols," Proceedings of the 15th International Information Security Conference, August 2000.

[18] Lemon, J. "Resisting SYN Flood DoS Attacks with a SYN Cache," USENIX BSDCon 2002 conference, San Francisco, CA.

[19] Li, J., Mirkovic, J., Wang, M., Reiher, P., Zhang, L., "SAVE: Source Address Validity Enforcement Protocol," INFOCOM 2002, June 2002.

[20] Mahajan, R., Bellovin, S., Floyd, S., Ioannidis, J., Paxson, V., Shenker, S., "Controlling High Bandwidth Aggregates in the Network," Technical report, (draft), February 2001.

[21] Mogul, J. and Ramakrishnan, K. K. "Eliminating Receive Livelock in an Interrupt-driven Kernel," Proceedings of the USENIX 1996 Annual Technical Conference, San Diego, CA.

[22] Moore, D., Voelker, G., and Savage, S., "Inferring Internet Denial of Service Activity," Proceedings of the 2001 USENIX Security Symposium, Washington D.C., August 2001.

[23] Postel, "Transmission Control Protocol," RFC 793.

[24] Savage, S., Cardwell, N., Wetherall, D., and Anderson, T., "TCP Congestion Control with a Misbehaving Receiver," ACM Computer Communications Review, pp. 71-78, v 29, no 5, October, 1999.

[25] Shreedhar, M., Varghese, G., "Efficient Fair Queueing Using Deficit Round Robin," SIGCOMM'95

[26] Snoren, A.C., Partridge, C., Sanchez, L.A., Jones, C.E., Tchakountio, F., Kent, S.T. Strayer, W.T., "Hash-Based IP Traceback," SIGCOMM 2001, August 2001.

[27] Spatscheck O. and Peterson, L., "Defending Against Denial of Service Attacks in Scout," Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation, pp 59-72, February 1999.

[28] Song, D.X. and Perrig, A., "Advanced and authenticated marking schemes for IP Traceback," INFOCOM 2001

[29] Stone, R., "CenterTrack: An IP Overlay Network for Tracking DoS Floods," Proceedings of 9th USENIX Security Symposium, August 2000.

[30] Vixie, P., Sneeringer, G., Schleifer, M., "Events of 21-Oct-2002." http://f.root-servers.org/october21.txt

EXHIBIT 2

Lucent Technologies

Request for Approval of Manuscript

BL03.00089

To:

PATENT

Suggested reviewer: Kenneth Brown

Date: February 14, 2003

Please review and return (or **FAX 908-582-2044**) before **March 7, 2003**.

for a talk intended for presentation at
ACM SIGCOMM 2003
Karlsruhe, Germany
August 25, 2003

intended for publication in
Proceedings, ACM SIGCOMM 2003

Title: Defense Against Distributed Denial of Service Attacks

Author	Company	Tel.	Loc.	Room	Org.
Lin, D.	LU	(908) 582-5744	100001	2C-419	10009639

Reviews:

The following individual(s) have been notified of the intention to release this material:
PATENT - Suggested Reviewer: Kenneth Brown

Additional Information:

The author's organization has answered the following questions as indicated:
* Is any of the work discussed supported by government funds? **no**

Abstract:

This paper presents a set of DDoS defense algorithms that can be efficiently implemented at network edges to reduce or eliminate the impact of bandwidth and state attacks on TCP servers.

Recommendation:

☐ **Approved**

SEND ALL COMMENTS DIRECTLY TO FIRST Lucent Technologies AUTHOR

☐ **ON HOLD until further notice:** I will discuss issues with author.

Patent Attorney — Date: 2/20/03

Please return to: Publication Clearance Service - Lucent Technologies
Room 2C-203, 600-700 Mountain Ave., Murray Hill, NJ 07974
MH 2C-203, (908) 582-2420

EXHIBIT 3

Class number: _____ NOV 0 1 2002

Criterion letter: _____ Date

Signature

For Dept. Head / Director's Use Only

For IP-Law Use Only

LUCENT TECHNOLOGIES, INC.

DISCLOSURE OF INVENTION

THIS DESCRIPTION SHOULD BE SUPPLEMENTED BY ATTACHING COPIES OF RELEVANT DOCUMENTS, SUCH AS TECHNICAL MEMORANDA, PUBLISHED OR TO-BE-PUBLISHED ARTICLES, AND ENGINEERING NOTEBOOK PAGES. (Also, if for any item there is insufficient space on the form, attach additional pages as necessary.)

1. DESCRIPTIVE TITLE OF THE INVENTION: Defense Against State Attacks on TCP Servers

INVENTOR #1: LIN, DONG
Name (Print)

Company/Location Lucent Technologies

Phone/E-mail 908-582-5744

Director's Name Eric H. Grosse

INVENTOR #2:
Name (Print)

Company/Location

Phone/E-mail

2. PRIMARY CONTACT
If more than one inventor is named above, who will have the primary responsibilities for interfacing with Lucent IP-Law with respect to preparing and prosecuting a patent application for the invention?

Inventor Name:

3. PRESENT STATE OF THE INVENTION

☐ Idea ☒ Research ☐ Development

☐ Manufacture (Product Name: Ship Date:)

4. GOVERNMENT CONTRACT INVENTION
Was the invention made under a government contact? ☐ Yes ☒ No

5. PRESENT STATE OF THE ART
Briefly describe the closest already-known technology that relates to the invention. This would include, for example, already existing products, methods or compositions which are known to you personally or through descriptions in publications or patents.

6. ADVANCEMENT IN STATE OF THE ART
Briefly describe the unique advancement achieved by the invention. This may be done, for example, by describing a problem with the prior art that is solved or specific objects that are achieved by the invention.

7. HOW ACHIEVED

Briefly describe the invention and how it achieves the advancement described in paragraph 6.

8. DISCLOSURE OUTSIDE OF LUCENT

Anticipated Publication Date: Jan 31, 2003

Publication Name: SIGCOMM

Submitted to Publication Clearance? ☐ Yes ☒ No

If the invention was or will otherwise be disclosed to any non-Lucent employee, describe to whom (person/company), when, where, why, and whether it was/will be under a non-disclosure agreement.

9. INVENTOR #1: _____ Oct 29, 2002

Signature Date

INVENTOR #2: _____

Signature Date

5 Present State of the Art

Each established TCP connection creates a state on a TCP server. The amount of allocated memory for storing these states is proportional to the total number of open connections. A denial-of-service attack on a TCP server is launched by continuously creating new TCP connections with the targeted server until it runs out of memory and unable to accept service requests from legitimate users.

There is no known defense against such DOS attack. Section 4.2.3.6 in RFC 1122 [1] addresses a related issue. A TCP keep-alive mechanism is invoked in server applications that might otherwise hang indefinitely and consume resources unnecessarily if a client crashes or aborts a connection during a network failure. This mechanism could be used to alleviate the TCP server during an attack in-progress.

However, the keep-alive mechanism can be easily defeated without much cost from the attacker. The attacking machines, usually victims of break-ins themselves, respond to keep-alive probes and effectively prevent the server from removing theirs states.

We can demonstrate that the memory required on the attacking host is constant and unrelated to the number of TCP connections created for the attack. We name this kind of DOS attack a "TCP stateless hog."

6 Advancement in State of the Art

See section 5 for a description of a DOS attack on the present state of the art.

This invention strengthens the TCP keep-alive mechanism and effectively detects and defeats a TCP stateless hog attack.

7 How Achieved

Among other information, a keep-alive probe packet contains a sequence number corresponding to the latest octet from the server to the client. The client responds with an acknowledgment in agreement with the same octet. The attacker answers the probe by copying the sequence number.

This invention proposes a random number to be used as the sequence number in the keep-alive probe packets. TCP's specification [2] requires that the acknowledgment segment contains the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received.

The construction of such acknowledgment requires maintaining states on the responding host as done in legitimate and conforming TCP implementations. We have tested and confirmed our idea against the TCP stacks in the following operating systems: Microsoft Windows, Linux, FreeBSD, OSF.

A TCP stateless hog program will produce the incorrect response and can be detected by the TCP server. TCP connections associated with these wrong answers can be removed with 100% accuracy.

[1] R. Braden, "Requirements for Internet Hosts – Communication Layers," RFC 1122.

[2] J. Postel, "Transmission Control Protocol," RFC 793.

EXHIBIT 4

INTELLECTUAL PROPERTY-LAW

Subject: **Patent Submission**
(IDS) No. 125553
Defense Against State Attacks
On TCP Servers - (CORRECTED TITLE)

date: **February 13, 2003**
from: **Kenneth M. Brown**
Corporate Counsel
Room 3B-519
(908) 582-5998
kmbrown1@lucent.com

To: Dong Lin

The above-identified patent submission, of which you appear to be an originator, was submitted recently to consider its patentability. IP-Law is committed to being as responsive as possible regarding each submission. You will be contacted as soon as a decision is made on whether to file a patent application based on this submission.

Each decision to proceed with a patent application is based on a careful weighing of the commercial value that the prospective patent could bring to Lucent through the intellectual property rights that it represents. Because due time must be allowed for this initial evaluation as well as for information gathering, draft preparation, and review, you should not be surprised if it takes twenty weeks or more before your patent application is ready for filing in the United States Patent and Trademark Office.

Patent applications are drafted by an assigned attorney. The assigned attorney may be an in-house Lucent attorney or an outside attorney retained by Lucent. If an outside attorney is assigned, I will be the Lucent *managing attorney*. In that role, I will supervise the preparation of the application, and I will be available for consultation as necessary. Currently, over half of our patent applications are assigned to outside attorneys.

Please notify me of any further developments regarding the nature of the invention or its uses. In addition, it is important to keep me informed of any related work by others, within or outside of Lucent, that might use this subject matter. Also, please notify me well in advance of any disclosure of the invention to anyone outside of Lucent.

The booklet, *A Quick Guide to The Patent Process At Lucent Technologies* may prove helpful. Copies can be ordered from my secretary, Cathy Dugan, at 908-582-7121, e-mail cdugan@lucent.com.

If you have any administrative questions, please also contact Cathy Dugan. For substantive issues regarding the submission, please feel free to contact me directly. Please refer to the IDS number noted above in any communications.

Kenneth M. Brown

KMB:cfd

cc: E.H. Grosse